Recircuit

Pablo Vázquez Zambrano
dpto. Ciencias de la Computación e
Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
pabvazzam@alum.us.es/pabvazzam@gmail.com

Resumen---Recircuit es una aplicación desarrollada en Python 2.7, la cual consiste en, dado un circuito reconfigurable donde existen puertas defectuosas, generar una configuración óptima del mismo de forma que dado unos bits de entrada que se le pasan al circuito, la salida que genere sea la correcta (o en su defecto lo más cercana posible a la que debería de salir). Para ello, se ha hecho usode un algoritmo genético por torneo.

Palabras Clave—Inteligencia Artificial, Algoritmo Genético, torneo, circuito, individuo, puerta.

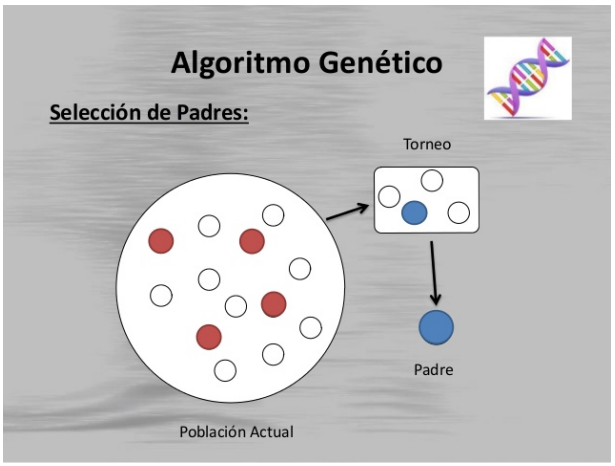

Figura 1

I. Introducción

En este proyecto se nos plantea que, dado un circuito de mxn (m capas con n puertas cada una) en el que cada puerta puede comportarse de diferente forma (AND, OR…) conectadas entre sí con dos inputs cada una como mucho, implementar un método de auto-reparación que dado el caso de haber sufrido daños active un proceso evolutivo que permita reconfigurarse para que pueda seguir comportándose como debería.

Para extender más el problema, se han añadido mejoras como puede ser meter dentro del problema dos tipos de puertas más que no se pedían, las cuales son las puertas NOR y XNOR. Además, con la ayuda de la librería Networkx [1] se dibujará un grafo que, aunque "desordenado", representará al circuito solución.

Para empezar, hemos creado una clase Python llamada "Puerta" que nos ayudará a representar a las diferentes puertas del circuito, que contarán con sus dos inputs provenientes de otra puerta (o de ninguna. En este caso contaría como "desactivada") y su output, calculado según los inputs y el tipo de puerta que sea.

Además de la clase Puerta, tendremos que definir las filas y columnas que compondrán nuestro circuito. Estos parámetros se pueden cambiar a nuestro antojo en el código.

Tras todo esto y por medio de diferentes métodos que nos permitirán realizar todo lo que necesitamos, se ejecutará el algoritmo genético y se creará una población de individuos (circuitos) con los valores de filas y columnas de puertas dados que, por medio de cruces y mutaciones, se elegirán a los n mejores de la población en función del fitness que obtengan, el cual vendrá dado por el mejor el menor, ya que será un problema de minimizar. Al finalizar cada generación, se mostrará el mayor y menor fitness de toda la población y la media, Aparte de mostrar el grafo por pantalla gracias a la librería de Matplotlib [2].

II. Preliminares

A. *Métodos empleados*

- Algoritmos genéticos [3]: Algoritmo que, mediante una población principal de individuos y operaciones de unión y mutación, trata de encontrar al mejor individuo.

B. *Trabajo relacionado*

Las referencias mencionadas a continuación no son trabajos en sí, pero nos han servido para realizar el proyecto:

- Ejemplos de algoritmo DEAP[4]: Esta librería, aunque nombrada posteriormente en la sección de las librerías, son ejemplos que vienen en la documentación de la misma, los cuales han servido para estructurar la aplicación y realizar los métodos necesarios para poder realizarla.

- Ayuda para definir el comportamiento de los distintos tipos de puerta[5].

C. *Librerías empleadas*

Mediante estas librerías hemos podido realizar satisfactoriamente el trabajo:

- Networkx [1]: Gracias a esta librería se ha podido elaborar un grafo de un circuito dado, el cual es direccional.
- Matplotlib [2]: Se usa para poder dibujar el grafo creado y poder mostrarlo en pantalla.
- Random: Esta librería de Python permite generar valores aleatorios, los cuales han sido muy importantes para poder elaborar los algoritmos de cruce, mutación y para crear una población inicial.
- Numpy: Consta de otra librería de Python con la que se han podido crear las estadísticas de mínimo, máximo y media de fitness de cada generación.
- Itertools: Gracias a esta librería de Python también, hemos podido realizar de una mejor forma el procedimiento de evaluar a un individuo, dado que nos facilita el poder obtener todas las combinaciones de bits para un número de los mismos dado.
- DEAP[4]: Es el centro de toda la aplicación, ya que mediante él se ha implementado el algoritmo genético. Consiste en una librería que, una vez dados ciertos parámetros como pueden ser las generaciones deseadas, el número de individuos en la población inicial, los métodos de mutación y unión, etc, te realiza el algoritmo.

III. METODOLOGÍA

A la hora de empezar, hemos tenido que estudiar cómo íbamos a representar lo que es la parte principal del problema: el circuito. Un mismo circuito, como hemos mencionado anteriormente, está compuesto de NxM puertas lógicas. Dado esto, debíamos pensar primero sobre las puertas.

Aquí nos encontramos la dificultad de que una puerta, teniendo en cuenta que puede tener distintas formas de comportamiento y diferentes inputs provenientes de otras puertas, no es algo que se pueda representar fácilmente mediante un carácter en general. Por ello, se decidió representar esta entidad mediante una clase Python, llamada Puerta, la cual tendrá varios atributos:

- puerta_entrada1: puerta conectada al primer de los inputs de la misma.
- puerta_entrada2: puerta conectada al segundo de los inputs de la misma.
- entrada1: valor de la salida de la puerta conectada al pirmer input (puerta_entrada1).
- entrada2: valor de la salida de la puerta conectada al segundo input (puerta_entrada2).
- salida: valor de salida de la misma puerta después de calcularla en función de los valores de entrada y el tipo de lógica de la puerta, atendiendo a las tablas lógicas. Si la puerta se considera como defectuosa, el valor siempre será 0.
- tipo: entero que va de 0 a 6. Cada valor representa una puerta lógica distinta, lo que indica la forma de comportarse a la hora de calcular la salida. Los tipos de puerta lógica que se pueden simular son los siguientes: AND, OR, NOT, NAND, XOR, NOR y XNOR.
- id: entero con valor de entre 0 y 1000 y aleatorio. Este valor nos permitirá diferenciar a las puertas unas de otras.

Cada puerta se representará mediante un par de valores (tipo, id), donde tipo se refiere al tipo de puerta lógica que simula e id al identificador de la misma.

Ya tenemos definida la estructura de las puertas, que son las que compondrán los diferentes circuitos. Dado esto, el siguiente paso será definir el circuito que, ya definidas las puertas, es fácil: cada circuito estará formado por mxn puertas.

Refiriéndonos a los términos de un algoritmo genético, podríamos decir que hasta ahora tenemos a los genes que compondrán un cromosoma y, a su vez, la estructura de cada cromosoma: los genes los compondrán las puertas, mientras que el cromosoma será un circuito, es decir, una población estará compuesta de individuos que a su vez son circuitos.

Ahora tenemos el problema para generar a un individuo. Esto es, generar un circuito aleatoriamente. El primer problema que había consistía en que dado como hemos planteado la composición de un individuo o cromosoma, no podemos generar los genes independientemente, sino que debemos generar a la vez el cromosoma y sus genes. Esto es, inicializar un circuito con unas filas y columnas dadas e ir rellenándolo con puertas cuyos atributos (tipo, puerta_entrada1 y puerta_entrada2) hayan sido generados aleatoriamente:

Procedimiento para generar un circuito aleatorio

Entrada:

Ninguna

Salida:

- El circuito generado

Pseudocódigo:

1. Se inicializa el circuito
2. Para i=0 hasta el número de filas
    a. Para j=0 hasta el número de columnas
        i. Si i=0
            1. Se crea una puerta que no tendrá puertas de entrada pero sí tipo de lógica
            2. Se asigna la puerta creada a la posición [i][j] del circuito
        ii. Si i=1
            1. Se crea una puerta cuyas puertas de entrada solo podrán ser de la primera fila del circuito (i=0) en caso de que por probabilidad se les añada
            2. Se asigna la puerta creada a la posición [i][j] del circuito
        iii. En cualquier otro caso
            1. Se crea una puerta cuyas puertas de entrada podrán ser de cualquiera de las dos filas inmediatamente inferiores del circuito (i=0) en caso de que por probabilidad se les añada
            2. Se asigna la puerta creada a la posición [i][j] del circuito
3. Devolver circuito creado

Pseudocódigo 1: Creación de un individuo o circuito

Antes de meternos de lleno en la implementación del algoritmo genético, nos queda un factor realmente importante y clave para este ejercicio, el cual consiste en, dada una lista de bits (0 o 1) de entrada, procesar la salida que resultaría tras haber recorrido todo el circuito.

Este apartado resulta realmente importante, ya que gracias a él podremos, mediante el algoritmo genético, estudiar cómo de "bueno" es un circuito, calculando su salida dadas las puertas defectuosas existentes y comparándola con la que debería de ser la correcta en caso de que sirva como solución (la salida que se obtendría con el circuito original en caso de que no haya ni una puerta defectuosa).

Procedimiento para calcular la salida dado un circuito

Entrada:

- Entrada que se le pasa en forma de lista
- Circuito al que le calcularemos la salida resultante
- Conjunto de puertas que resultan estar defectuosas

Salida:

- Salida resultante

Pseudocódigo:

1. Se inicializa la salida
2. Para i=0 hasta el número de filas
    a. Para j=0 hasta el número de columnas
        i. Si i=0
            1. Si la puerta está defectuosa
                a. La salida de la puerta en [i][j] del circuito es 0
            2. Si no
                a. Se le asigna a la primera entrada el valor de entrada[j] y se calcula la salida resultante, dando por hecho de que la segunda entrada resulta ser 0
        ii. Si i=filas-1
            1. Si la puerta está defectuosa
                a. La salida de la puerta en [i][j] del circuito es 0, y por tanto lo es en salida[j]
            2. Si no

a. Se calcula el valor de salida de la puerta en función de sus valores de entrada y se asigna ese valor a salida[j]

iii. En cualquier otro caso

1. Si la puerta está defectuosa

a. La salida de la puerta en [i][j] del circuito es 0

2. Si no

a. Se calcula el valor de salida de la puerta en función de sus valores de entrada

3. Se devuelve la salida entera

Pseudocódigo 2: Cálculo de la salida de un circuito dada una entrada

Una vez tenemos esta parte clave, podemos entrar en el algoritmo genético.

Al dar comienzo el algoritmo, se crea la población inicial y tras varios procesos de unión y mutación se van generando más individuos diferentes.

El proceso de unión consiste en que, dado dos individuos, uno adquiere propiedades (o características del otro) y viceversa. Dicho esto, en nuestro caso debería de trabajar de forma que en nuestro problema, dados dos circuitos, se intercambien las puertas en ciertas posiciones.

Sin embargo, al ser las puertas un tipo de clase-objeto hablando técnicamente desde el lenguaje de la programación, llegaba a dar problemas a la hora de realizar este proceso, por lo que se ha simplificado y en vez de cambiar todas las características de las puertas, se ha implementado de forma que solamente se cambien el tipo de lógica.

Método de cruce del algoritmo genético

Entrada:

- Individuo1
- Individuo2

Salida:

- Individuo1 con el tipo de lógica que tenía el individuo2
- Individuo2 con el tipo de lógica que tenía el individuo1

Pseudocódigo:

1. Para i=0 hasta filas*columnas/4

a. Se eligen una “fila” y “columna” aleatorios

b. Las puertas que se encuentran en la posición [fila][columna] de cada circuito se intercambian los tipos de lógica

Se devuelven los dos individuos modificados

Pseudocódigo 3: Proceso de cruce de dos individuos

Como se puede ver, se ha definido una especie de ecuación a la hora de calcular cuántas puertas serán las que se intercambien las lógicas. Esto es, porque si ponemos una cantidad de puertas fijas para cruzarse, podría ser o muy pocas o muchas dependiendo del tamaño de los circuitos. Por tanto, se ha optado por el hecho de que se cambien el número de puertas resultante del resultado de multiplicar las filas y columnas del circuito y dividirlo entre 4.

Ya que hemos hablado del proceso de cruce, toca hablar del proceso de mutación. En nuestro caso, iremos recorriendo todo el circuito a mutar y dado un índice de probabilidad, en caso de que resulte favorable la mutación mediante por probabilidad, se modificará completamente esa puerta.

Método de cruce del algoritmo genético

Entrada:

- Individuo a mutar
- Índice de probabilidad "indpb"

Salida:

- Individuo mutado

Pseudocódigo:

1. Para i=0 hasta el número de filas
    a. Para j=0 hasta el número de filas
        i. Si un número decimal aleatorio entre 0 y 1 es menor que indpb
            1. Se muta la puerta dadas unas restricciones en función de la fila en la que se encuentre la misma

Se devuelve el individuo mutado

Pseudocódigo 4: Proceso de mutación de un individuo

Habiendo hablado de estos dos procesos, queda hablar de otra parte importante de la que se compone un algoritmo genético, y es la función Fitness.

Esta función es la que nos ayuda a evaluar cómo de bueno es un individuo. Puede consistir tanto en maximizar un valor como en minimizar, y en este caso definiremos que el mejor individuo será aquel que tenga 0 errores en la salida, es decir, a menos errores tenga mejor será el individuo, por lo que el problema consistirá en minimizar el número de errores.

Primero calcularemos la salida resultante para todas las posibles combinaciones de un tamaño de bits (si el circuito contiene 5 columnas, consistirá en todas las combinaciones posibles con 5 bits) para el circuito a evaluar dadas las puertas defectuosas que existen y para el circuito original simulando su perfecto funcionamiento, y posteriormente se compararán las salidas y cuantos más bits distintos tengan en las diferentes posiciones del conjunto de bits, mayor será la penalización y por tanto peor calificación tendrá a la hora de ser elegido.

Función Fitness

Entrada:

- Individuo a calificar

Salida:

- Fitness resultante

Pseudocódigo:

1. Se inicializa una variable double "errores" a 0.0
2. Para "permutacion" en el conjunto de combinaciones posibles de bits de entrada
    a. Se calculan las salidas
        i. Para i hasta el tamaño de bits de salida
            1. Si los bits no son iguales se suma a la variable "errores" 10.0
3. Se define la variable "fitness" como el valor resultante de 1000.0*errores

Devolver fitness

Pseudocódigo 5: Proceso de calificación de un individuo

Ya teniendo todas las herramientas para implementar adecuadamente el algoritmo genético para dar solución al problema original, podemos definir la estructura completa.

En primer lugar se elaborarán una cierta cantidad de individuos aleatorios. Tras varias series de mutación y de cruce entre unos y otros se calculará el Fitness de cada individuo y pasarán a la siguiente generación.

Hay que recordar que es un algoritmo genético por torneo. Esto es, que a la hora de elegir el individuo para el proceso de union se elegirá una cantidad n de individuos, de los cuales el que tenga una mejor puntuación será el candidato. Tras x generaciones, el mejor individuo (aquel con mejor puntuación) será el elegido.

Finalmente, el problema planteado se ha implementado de la siguiente forma: dado un circuito, se comprueba mediante el procedimiento de cálculo de salida si hay algún problema desde el punto de vista de esta última, es decir, que la salida no sea la esperada. En ese caso, se procederá a ejecutar el algoritmo genético para encontrar una configuración del circuito que se pueda sustituir por la que ya tenia, al ser reconfigurable.

Hay que destacar por tanto que solo existirá solución si tras el algoritmo, el elegido mejor individuo contiene 0 errores o lo que es lo mismo, que su Fitness sea 0. En cualquier otro caso, seguirá habiendo errores y por tanto no tendría sentido la solución.

Además de que un algoritmo genético no siempre encontrará una solución óptima debido a que funciona por probabilidad, en nuestro caso habrá posibles casos en los que será imposible o casi imposible encontrar solución. Por ejemplo, si se nos estropea alguna puerta que esté directamente conectada con la salida, el valor de salida en esa posición será siempre 0 y por tanto será prácticamente imposible encontrar una solución.

IV. MEJORAS INTRODUCIDAS

Con el fin de enriquecer la solución, se han realizado una serie de mejoras aparte de los requisitos especificados:

- Representación gráfica de los circuitos, tanto del que partiremos como de el/los circuito/s solución.
- Añadido de más tipos de puertas lógicas al circuito, para que haya una mayor variedad dentro de nuestro algoritmo. Concretamente, se han añadido las puertas lógicas NOR y XNOR.

V. ANÁLISIS DE RENDIMIENTO

Para analizar mejor la solución implementada, recogeremos datos cambiando el tamaño del circuito, además del número de individuos por población y de generaciones del algoritmo.

Para hacer un análisis lo más linear posible, simularemos el caso de que se estropean la mitad de las filas de los circuitos. Además, también ajustaremos para que la selección por torneo sea siempre de 5 individuos.

- **NxM**: Filas y columnas del circuito.
- **Población**: Cantidad de la población inicial.
- **Generaciones**: Número de generaciones.
- **Tiempo**: Tiempo que ha tardado el algoritmo.
- **Acierto**: Si ha encontrado sustituto o no.

NxM	Población	Generaciones	Tiempo	Acierto
3x3	20	20	<1s	No
3x3	20	50	<1s	No
3x3	20	100	~1s	Sí
3x3	30	20	<1s	No
3x3	30	50	<1s	Sí
3x3	30	100	~1s	Sí
5x5	20	20	<1s	Sí
5x5	20	50	~2s	Sí
5x5	20	100	~4s	Sí
5x5	30	20	~1s	No
5x5	30	50	~3s	Sí
5x5	30	100	~6s	Sí
8x8	20	20	~10s	No
8x8	20	50	~30s	No
8x8	20	100	~49s	Sí
8x8	30	20	~15s	No
8x8	30	50	~40s	Sí
8x8	30	100	~80s	Sí

El proceso para realizar cada prueba ha sido el siguiente:

Se crea un circuito aleatorio del tamaño dado y se ejecuta el algoritmo genético sobre él. Esto, repetidas 5 veces, dada la aleatoriedad del mismo, se ha puesto en la columna "**Acierto**" el que ha salido más veces.

Dicho esto, cabe destacar que en las pruebas con mayor número de generaciones y de población no han habido (prácticamente ningún) caso en el que el resultado sea negativo.

VI. CONCLUSIÓN

A la hora de analizar los resultados, hay que entender que, aunque la función del algoritmo genético pretende dar soluciones a varios problemas planteados de una forma parecida, en él juega un papel esencial la aleatoriedad.

Con esto se quiere decir que en muchos casos puede ser que el resultado sea improbable que llegue a ser exitoso u óptimo o incluso que sea imposible.

En nuestro caso, alguna ocasión en la que sea imposible encontrar una solución óptima puede ser el ya mencionado, donde se corrompen todas las puertas de la última fila. De hecho, por ejemplo, el número de puertas (con respecto al tamaño del circuito) tiene una gran importancia dentro de nuestro algoritmo, ya que por ejemplo no es lo mismo que se rompan 5 puertas en un tablero de 3x3 que se rompan el mismo número de puertas en un tablero de 10x10. Claro está que puede ser también que estas puertas que se convierten en defectuosas estén situadas en posiciones críticas del circuito, lo que dificultarían la búsqueda de un resultado óptimo o puede que incluso la haga imposible.

También cabe destacar, analizando nuestra tabla de pruebas, que resulta mucho más probable que acierte el algoritmo cuanto mayor sea el número de la población inicial y las generaciones. Sin embargo, tiene un precio debido a que, cuanto mayores sean estos parámetros, mayor es el tiempo y los recursos que el sistema necesita para ejecutarlo.

Finalmente, podemos ver que nuestro algoritmo resulta ser consistente y útil a costa de consumir bastantes recursos, ya que no presenta errores teniendo unos parámetros de población inicial y número de generaciones relativamente grandes.

VII. Especificaciones

La realización de tanto la implementación como el análisis del algoritmo se han realizado con un equipo con procesador Intel Core I7-7700K 4.2GHz y una tarjeta RAM DDR4de 16GB y CL15.

VIII. Referncias

[1] Networkx https://networkx.github.io/

[2] Matplotlib https://matplotlib.org/

[3] Página web de la asignatura de Inteligencia Artificial https://www.cs.us.es/cursos/iais

[4] Deap https://deap.readthedocs.io/en/master/

[5] Ayuda para definir el comportamiento de los distintos tipos de puerta https://es.wikipedia.org/wiki/Puerta_l%C3%B3gica